Information Processing Letters 116 (2016) 71–74

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A filtration method for order-preserving matching

Tamanna Chhabra, Jorma Tarhio *

Department of Computer Science, Aalto University, P.O. Box 15400, FI-00076 Aalto, Finland

ARTICLE INFO

Article history:
Received 5 March 2015
Received in revised form 16 September 2015
Accepted 19 October 2015
Available online 2 November 2015
Communicated by Tsan-sheng Hsu

Keywords:
Algorithms
Combinatorial problems
Order-preserving matching
String searching

ABSTRACT

The problem of order-preserving matching has gained attention lately. The text and the pattern consist of numbers. The task is to find all the substrings in the text which have the same length and relative order as the pattern. The problem has applications in analysis of time series. We present a new sublinear solution based on filtration. Any algorithm for exact string matching can be used as a filtering method. If the filtration algorithm is sublinear, the total method is sublinear on average. We show by practical experiments that the new solution is more efficient than earlier algorithms.

1. Introduction

String matching [1] is a widely known problem in Computer Science. Given a text T of length n and a pattern P of length m, both being strings over a finite alphabet Σ, the task of string matching is to find all the occurrences of P in T. The problem of order-preserving matching [2–6] has gained attention lately. It considers strings of numbers. The task is to find all the substrings (also called factors) u in T which have the same relative order as P, and $|u| = |P|$. Suppose $P = (10, 22, 15, 30, 20, 18, 27)$ and $T = (22, 85, 79, 24, 42, 27, 62, 40, 32, 47, 69, 55, 25)$, then the relative order of P matches the substring $u = (24, 42, 27, 62, 40, 32, 47)$ of T, see Fig. 1.

Several online [7,5,3,4] and one offline solution [2] have been proposed for order-preserving matching. Kubica et al. [4] and Kim et al. [3] presented solutions based on the Knuth–Morris–Pratt algorithm (KMP) [8]. Later, Cho et al. [5,6] gave a sublinear solution based on the bad character heuristic of the Boyer–Moore algorithm [9]. Almost at the same time, Belazzougui et al. [7] derived an optimal sublinear solution. We will present a new practical solution based on filtration. We form a modified pattern and use an algorithm for exact string matching as a filtration method. Our approach is simpler and in practice more efficient than earlier solutions. We transform the original pattern P into a binary string P' expressing increases (1), equalities (0), and decreases (0) between subsequent pattern positions. Then we search for P' in the analogously transformed text T'. For example, $P' = 101001$ corresponds to $P = (10, 22, 15, 30, 20, 18, 27)$ and $T' = 100101001100$ to T above. Each occurrence is a match candidate which is verified following the numerical order of the positions of the original pattern P. Note that in this approach any algorithm for exact string matching can be used as a filtration method. If the filtration algorithm is sublinear and the text is transformed on line, the total method is sublinear on average.

We made experiments with two sublinear string matching algorithms and two linear string matching algorithms as the filtering method. Our approach with sublinear filters was considerably faster than the algorithm by Cho et al. [5], which is the first sublinear solution of the problem.

* Corresponding author.
E-mail addresses: tamanna.chhabra@aalto.fi (T. Chhabra), jorma.tarhio@aalto.fi (J. Tarhio).

http://dx.doi.org/10.1016/j.ipl.2015.10.005

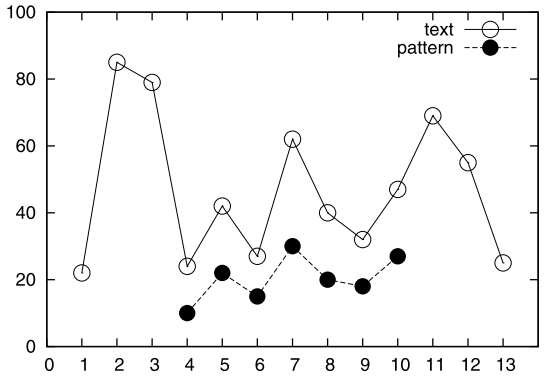

Fig. 1. Example of order-preserving matching.

The paper is organized as follows. Section 2 describes the previous solutions for order-preserving matching, Section 3 presents our solution based on filtration, Section 4 analyses the new approach, Section 5 presents and discusses the results of practical experiments, and Section 6 concludes the article.

2. Previous solutions

In the first KMP approach presented by Kubica et al. [4], the fail function in the KMP algorithm is modified to compute the order-borders table. This can be achieved in linear time. The KMP algorithm is mutated such that it determines if the text contains substring with the same relative order as that of the pattern using the order-borders table. This computation can be done in linear time. Hence, the total time complexity of the method is linear.

The second KMP approach by Kim et al. [3] is based on the prefix representation. The prefix representation is based on finding the rank of each number in the prefix. The time complexity of the method is $O(n \log m)$. This approach is further optimized using the nearest neighbor representation to overcome the overhead involved in computing the rank function. The time complexity of the improved version is $O(n + m \log m)$.

The BMH approach by Cho et al. [5] is based on the bad character rule applied to q-grams, i.e. strings of q characters. A q-gram is treated as a single character in order to make shifts longer. In this way, a large amount of text can be skipped for long patterns, and the algorithm is sublinear on average. The standard version works in $O(mn)$ in the worst case. Later, Cho et al. [6] introduced a linear version, which has been combined with KMP in order to guarantee linear behavior in the worst case.

3. Our solution

In Section 1 we gave an informal description of order-preserving matching. Let us define the problem formally.

Problem definition Two strings $u = u_1 u_2 \cdots u_m$ and $v = v_1 v_2 \cdots v_m$ of the same length over Σ are called *order-isomorphic* [3,4], written $u \approx v$, if

$$u_i \leq u_j \Leftrightarrow v_i \leq v_j \text{ for } 1 \leq i, j \leq m.$$

In the *order-preserving pattern matching problem*, the task is to find all the substrings of $T = t_1 t_2 \cdots t_n$ which are order-isomorphic with $P = p_1 p_2 \cdots p_m$.

Our solution for order-preserving matching consists of two phases: filtration and verification. First the text is transformed to a bit string which is filtered with some exact string matching algorithm. In the second phase the match candidates are verified using a checking routine.

Filtration For filtration, the consecutive numbers in the pattern $P = p_1 p_2 \cdots p_m$ are compared pairwise in the preprocessing phase and the result is encoded as a modified pattern $P' = b_1 b_2 \cdots b_{m-1}$ of binary numbers: b_i is 1 if $p_i < p_{i+1}$ holds, otherwise b_i is 0. In the search phase, some algorithm for exact string matching (let us call it A) is applied to filter out the text. When Algorithm A reads an alignment window of the original text, the text is encoded incrementally online in the same way as the pattern. Algorithm A is run as if the whole text would have been encoded. Because Algorithm A may recognize an occurrence of P' which does not correspond to an actual match of P in T, each occurrence of P' is only a match candidate which should be verified. It is clear that this filtration method cannot skip any occurrence of P in T.

Verification During preprocessing the pattern, the numbers of the pattern $P = p_1 p_2 \cdots p_m$ are sorted. The result is an auxiliary table r: $p_{r[i]} \leq p_{r[j]}$ holds for each pair $i < j$ and $p_{r[1]}$ is the smallest number in P. In addition, we need a binary vector E representing the equalities: $E[i] = 1$ denotes that $p_{r[i]} = p_{r[i+1]}$ holds. The match candidates found by Algorithm A are traversed in accordance with the table r. If the candidate starts from t_j in T, the first comparison is done between $t_{j-1+r[1]}$ and $t_{j-1+r[2]}$. There is a mismatch when

$t_{j-1+r[i]} > t_{j-1+r[i+1]}$ or

$(t_{j-1+r[i]} = t_{j-1+r[i+1]}$ and $E[i] = 0)$ or

$(t_{j-1+r[i]} < t_{j-1+r[i+1]}$ and $E[i] = 1)$

is satisfied. The candidate is discarded when a mismatch is encountered. Verification is efficient because sorting is done only once during preprocessing.

Remark We use binary numbers in encoding. We also tried encoding of three numbers 0, 1, and 2 corresponding to '<', '=', and '>', but the binary approach was faster in practice, because testing of one condition is faster than testing of two conditions. Also the frequency of nearby equalities is low in real data.

4. Analysis

We will prove that our approach is sublinear in the average case, if the filtration algorithm is sublinear. Sublinearity means that on average all the characters in the text are not examined.

Let us assume that the numbers in P and T are integers and they are statistically independent of each other and the distribution of numbers is discrete uniform. Let